

# Well-typed programs can't be blamed

Philip Wadler

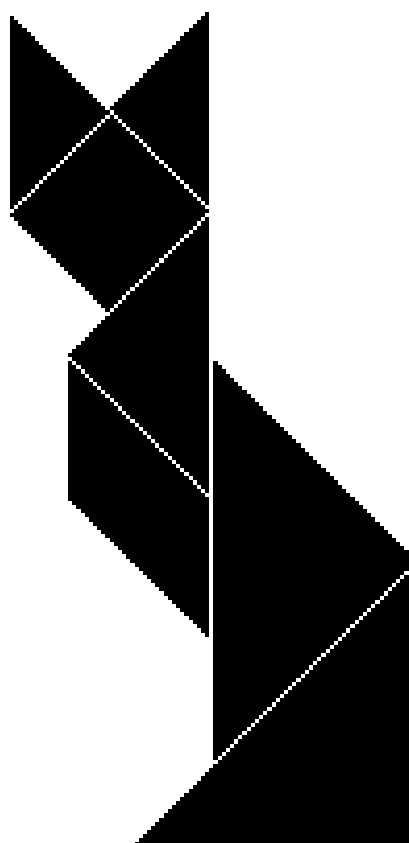
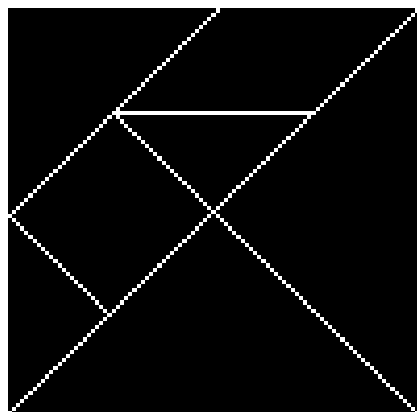
University of Edinburgh

Robert Bruce Findler

University of Chicago







# A repeated theme

Findler and Felleisen (2002):

Contracts

Flanagan (2006):

Hybrid types

Siek and Taha (2006):

Gradual types

Evolving a program

## An untyped program

[let

$x = 2$

$f = \lambda y. y + 1$

$h = \lambda g. g (g x)$

in

$h f]$

$\longrightarrow$

[4]

## A typed program

let

$x = 2$

$f = \lambda y : \text{Int}. y + 1$

$h = \lambda g : \text{Int} \rightarrow \text{Int}. g (g x)$

in

$h f$

$\longrightarrow$

$4 : \text{Int}$



## A partly typed program—narrowing

let

$x = 2$

$f = \langle \text{Int} \rightarrow \text{Int} \Leftarrow \text{Dyn} \rangle^p [\lambda y. y + 1]$

$h = \lambda g : \text{Int} \rightarrow \text{Int}. g (g x)$

in

$h f$

$\longrightarrow$

$4 : \text{Int}$

## A partly typed program—narrowing

```
let
  x = 2
  f =  $\langle \text{Int} \rightarrow \text{Int} \Leftarrow \text{Dyn} \rangle^p [\lambda y. 'b']$ 
  h =  $\lambda g : \text{Int} \rightarrow \text{Int}. g (g x)$ 
in
  h f
→
  blame p
```

Positive (covariant): blame the term contained in the cast

## Another partly typed program—widening

let

$x = [2]$

$f = \langle \text{Dyn} \Leftarrow \text{Int} \rightarrow \text{Int} \rangle^p (\lambda y : \text{Int}. y + 1)$

$h = [\lambda g. g (g x)]$

in

$[h f]$

$\longrightarrow$

$[4]$

## Another partly typed program—widening

```
let
   $x = \lceil 'a' \rceil$ 
   $f = \langle \text{Dyn} \Leftarrow \text{Int} \rightarrow \text{Int} \rangle^p (\lambda y : \text{Int}. y + 1)$ 
   $h = \lceil \lambda g. g (g x) \rceil$ 
in
   $\lceil h f \rceil$ 
 $\longrightarrow$ 
  blame  $\bar{p}$ 
```

Negative (contravariant): blame the context containing the cast

# The Blame Game

# Blame

$\langle \text{Int} \Leftarrow \text{Dyn} \rangle^p [2]$

$\longrightarrow$

2

$\langle \text{Int} \Leftarrow \text{Dyn} \rangle^p ['a']$

$\longrightarrow$

blame  $p$

## The Blame Game—widening

$(\langle \text{Dyn} \rightarrow \text{Dyn} \Leftarrow \text{Int} \rightarrow \text{Int} \rangle^p (\lambda y : \text{Int}. y + 1)) \text{ [2]}$

$\longrightarrow$

$\langle \text{Dyn} \Leftarrow \text{Int} \rangle^p ((\lambda y : \text{Int}. y + 1) (\langle \text{Int} \Leftarrow \text{Dyn} \rangle^{\bar{p}} \text{ [2]}))$

$\longrightarrow$

$\text{[3]}$

## The Blame Game—widening

$(\langle \text{Dyn} \rightarrow \text{Dyn} \Leftarrow \text{Int} \rightarrow \text{Int} \rangle^p (\lambda y : \text{Int}. y + 1)) \text{ [ ' a ' ]}$

$\longrightarrow$

$\langle \text{Dyn} \Leftarrow \text{Int} \rangle^p ((\lambda y : \text{Int}. y + 1) (\langle \text{Int} \Leftarrow \text{Dyn} \rangle^{\bar{p}} \text{ [ ' a ' ]}))$

$\longrightarrow$

$\text{blame } \bar{p}$

Widening can give rise to negative blame, but never positive blame



## The Blame Game—narrowing

$$(\langle \text{Int} \rightarrow \text{Int} \Leftarrow \text{Dyn} \rightarrow \text{Dyn} \rangle^p (\lambda y : \text{Dyn}. \lceil y + 1 \rceil)) \ 2$$

→

$$\langle \text{Int} \Leftarrow \text{Dyn} \rangle^p ((\lambda y : \text{Dyn}. \lceil y + 1 \rceil) (\langle \text{Dyn} \Leftarrow \text{Int} \rangle^{\bar{p}} 2))$$

→

3

## The Blame Game—narrowing

$$\begin{aligned} & (\langle \text{Int} \rightarrow \text{Int} \Leftarrow \text{Dyn} \rightarrow \text{Dyn} \rangle^p (\lambda y : \text{Dyn}. [\text{' b ' }])) \text{ 2} \\ \longrightarrow & \\ & \langle \text{Int} \Leftarrow \text{Dyn} \rangle^p ((\lambda y : \text{Dyn}. [\text{' b ' }])) (\langle \text{Dyn} \Leftarrow \text{Int} \rangle^{\bar{p}} \text{ 2})) \\ \longrightarrow & \\ & \text{blame } p \end{aligned}$$

Narrowing can give rise to positive blame, but never negative blame

And now a word from our sponsor

JOURNAL OF  
*Functional Programming*

VOLUME 10 PART 2 SEPTEMBER 2001



CAMBRIDGE  
UNIVERSITY PRESS





Untyped and supertyped

# Untyped = Uni-typed

$$\llbracket x \rrbracket = x$$

$$\llbracket n \rrbracket = \langle \text{Dyn} \Leftarrow \text{Int} \rangle n$$

$$\llbracket \lambda x. N \rrbracket = \langle \text{Dyn} \Leftarrow \text{Dyn} \rightarrow \text{Dyn} \rangle (\lambda x : \text{Dyn}. \llbracket N \rrbracket)$$

$$\llbracket L M \rrbracket = (\langle \text{Dyn} \rightarrow \text{Dyn} \Leftarrow \text{Dyn} \rangle \llbracket L \rrbracket) \llbracket M \rrbracket$$

(slogan due to Bob Harper)



# Contracts

$$\text{Nat} = \{x : \text{Int} \mid x \geq 0\}$$

let

$$x = \langle \text{Nat} \Leftarrow \text{Int} \rangle 2$$

$$f = \langle \text{Nat} \rightarrow \text{Nat} \Leftarrow \text{Int} \rightarrow \text{Int} \rangle (\lambda y : \text{Int}. y + 1)$$

$$h = \lambda g : \text{Nat} \rightarrow \text{Nat}. g (g x)$$

in

$$h f$$

$\longrightarrow$

$$4_{\text{Nat}} : \text{Nat}$$

# Subtyping

# Subtype

$$\overline{\text{Dyn} <: \text{Dyn}}$$

$$\frac{S' <: S \quad T <: T'}{S \rightarrow T <: S' \rightarrow T'}$$

$$\frac{s \text{ implies } t}{\{x : B \mid s\} <: \{x : B \mid t\}}$$

Example:

$$\text{Dyn} \rightarrow \text{Int} <: \text{Int} \rightarrow \text{Dyn}$$

$$\text{Int} \rightarrow \text{Nat} <: \text{Nat} \rightarrow \text{Int}$$

## Positive subtype—widening

$$\overline{S <: ^+ \text{Dyn}}$$

$$\frac{S' <: ^- S \quad T <: ^+ T'}{S \rightarrow T <: ^+ S' \rightarrow T'}$$

$$\frac{s \text{ implies } t}{\{x : B \mid s\} <: ^+ \{x : B \mid t\}}$$

Examples:

$$\text{Int} \rightarrow \text{Int} <: ^+ \text{Dyn} \rightarrow \text{Dyn}$$

$$\text{Nat} \rightarrow \text{Nat} <: ^+ \text{Int} \rightarrow \text{Int}$$

## Negative subtype—narrowing

$$\overline{\text{Dyn} <:^- T}$$

$$\frac{S' <:^+ S \quad T <:^- T'}{S \rightarrow T <:^- S' \rightarrow T'}$$

$$\overline{\{x : B \mid s\} <:^- \{x : B \mid t\}}$$

Examples:

$$\text{Dyn} \rightarrow \text{Dyn} <:^- \text{Int} \rightarrow \text{Int}$$

$$\text{Int} \rightarrow \text{Int} <:^- \text{Nat} \rightarrow \text{Nat}$$

# Naive subtype

$$\overline{S <:_n \text{Dyn}}$$

$$\frac{S <:_n S' \quad T <:_n T'}{S \rightarrow T <:_n S' \rightarrow T'}$$

$$\frac{s \text{ implies } t}{\{x : B \mid s\} <:_n \{x : B \mid t\}}$$

Example:

$$\text{Int} \rightarrow \text{Int} <:_n \text{Dyn} \rightarrow \text{Dyn}$$

$$\text{Nat} \rightarrow \text{Nat} <:_n \text{Int} \rightarrow \text{Int}$$

And now . . . a theorem!

# The Blame Theorem

Consider a source program, where  $p$  appears only once.

- If  $S <:^+ T$  then  $\langle T \Leftarrow S \rangle^p_s \not\rightarrow \text{blame } p$ .
- If  $S <:^- T$  then  $\langle T \Leftarrow S \rangle^p_s \not\rightarrow \text{blame } \bar{p}$ .



## The Blame Lemma

Let  $t$  be a well-typed term and  $p$  be a blame label, and consider all subterms of  $t$  containing  $p$ . If

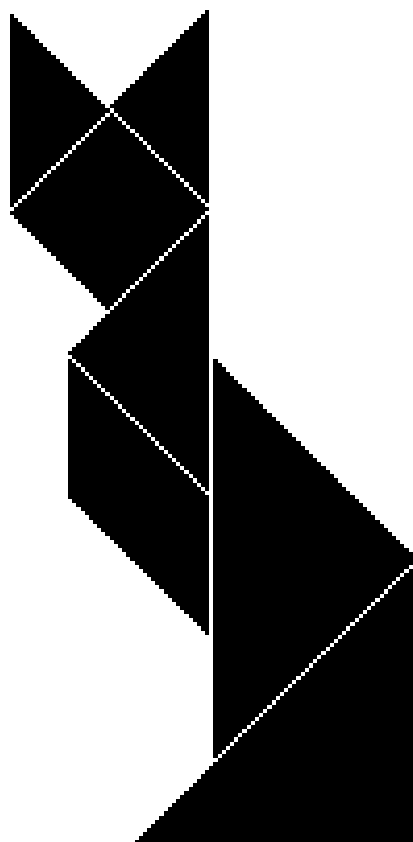
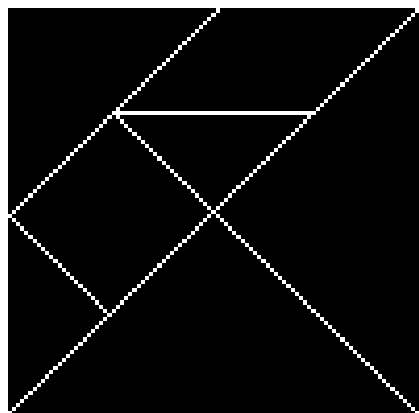
- every cast with label  $p$  is a positive subtype,

$$\langle T \Leftarrow S \rangle^p s \text{ has } S <:^+ T$$

- every cast with label  $\bar{p}$  is a negative subtype,

$$\langle T \Leftarrow S \rangle^{\bar{p}} s \text{ has } S <:^- T$$

then  $t \not\rightarrow^* \text{blame } p$ .



# The First Tangram Theorem

$S <: T$  if and only if  $S <:^+ T$  and  $S <:^- T$

## The Blame Corollary

Consider a source program, where  $p$  appears only once.

- If  $S <: T$  then  $\langle T \Leftarrow S \rangle^{p_s} \not\rightarrow \text{blame } p, \text{blame } \bar{p}$ .

# The Second Tangram Theorem

$S <:_n T$  if and only if  $S <:^+ T$  and  $T <:^- S$

## The Blame Corollaries

Consider a source program, where  $p$  appears only once.

- If  $S <:_n T$  then  $\langle T \Leftarrow S \rangle^{p_s} \not\rightarrow \text{blame } p$ .
- If  $T <:_n S$  then  $\langle T \Leftarrow S \rangle^{p_s} \not\rightarrow \text{blame } \bar{p}$ .

# Conclusion

# A new slogan for type safety

Milner (1978):

Well-typed programs can't go wrong.

Harper; Felleisen and Wright (1994):

Well-typed programs don't get stuck.

Wadler and Findler (2008):

Well-typed programs can't be blamed.